Roombots : Toward Emancipation of Furniture. A Kinematics-Dependent Reconfiguration Algorithm for Chain-Type Modular Robots

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

Sébastien GAY

Department of Computer Science

Institute of applied sciences (INSA) Lyon

A thesis submitted for the degree of

Master of Philosophy

Yet to be decided

To whom it may concern ...

Acknowledgements

FIRST of all, I would like to acknowledge my Master's project advisor, Pr. Auke Ijspeert for having welcomed me in his laboratory. I am grateful that he proposed me this interesting project and found funding for me, although it is not a usual occurrence for a Master's student. I would like to thank Dr. Masoud Asadpour, who worked on the same project as me and helped me tremendously.

I would like to thank Dr. Jonas Buchli and again Pr. Auke Ijspeert and Dr. Masoud Asadpour for they very pertinent comments on my Master thesis and defense.

I am also thankful to my school, INSA Lyon, where I have spent these last five years which were very enriching. I would especially like to thank Dr. Guillaume Beslon for the quality of his instruction, which motivated me to do robotics and for having guided me toward the EPFL.

I would like to thank all the people from the Biologically Inspired Robotic Group for their very friendly welcome and for the very serious but kind atmosphere that we had in the laboratory.

I had thoroughly a great experience, both professionally and humanly, during this project at the BIRG and would like to acknowledge everyone that contributed to it and that I may have forgotten in the previous list.

Abstract

MODULAR robots are complex systems composed of a set of simple robotic units. These units can rearrange to change the shape of the whole structure. This ability enables the reconfigurable robots to be extremely adaptive to changes in the environment and task. The most common applications of these systems are in space exploration where their versatility is a big advantage.

Our project is very distant from this field. The aim of the project is to use modular robot units to build furniture that could change shape and locomote in various environments.

This document tackles the problem of reconfiguration of a set of modular units : basically, what are the necessary connections and disconnections to reconfigure one shape to another ? This problem can be viewed as a graph theory problem, but in this case, we do not take into account the kinematical restrictions of the modules. Thus, the sequences found are not necessarily feasible on the real system. We introduce kinematical information in our model in order to avoid internal collisions and disconnection of the structure. This enables us to generate the tree of *valid moves*.

However, the size of the set of movements that we have to check to build this tree grows exponentially with the number of modules. We present the *Module Distance Metric* and use it to decrease the size of the set of candidate solutions. We then present a hashing method to delete duplicates in this search space.

We express the size of the search space theoretically and show that it is decreased significantly by our methods. We make some computational time measurements and show that the results are in accord with the theoretical study. We then expose some interesting sequences found by our algorithm and implement them into real robots thus concretising our approach.

Résumé

LES robots modulaires sont des systèmes complexes composés d'un ensemble d'unités robotiques simples. Ces unités peuvent se réarranger pour modifier la forme de toute la structure. Cette capacité permet aux robots reconfigurables d'être extrêmement adaptatifs aux changements d'environnement ou de tâche. Les applications les plus courantes de ces systèmes sont en exploration spatiale où leur polyvalence est un gros avantage.

Notre projet est très distant de ce domaine. Ce que nous visons est d'utiliser des modules robotique pour construire des meubles qui puissent changer de forme et se déplacer dans des environnements variés.

Ce document traite le problème de la reconfiguration d'un ensemble de modules : quelles sont les connections et déconnections nécessaires pour passer d'une forme à une autre ? Ce problème peut être vu comme un problème de théorie des graphes mais ne prends alors pas en compte les restrictions cinématiques des modules. Par conséquent les séquences trouvées ne sont pas nécessairement faisables sur le systéme réel. Nous introduisons ces informations cinématiques dans notre modèle pour éviter les collisions internes, les déconnections de la structure, et générons l'arbre de tous les *mouvements valides*.

Mais la taille de l'ensemble des mouvements que nous devons valider pour construire cet arbre augmente de façon exponentielle avec le nombre de modules. Nous présentons le *Module Distance Metric* et l'utilisons pour réduire la taille de l'espace des solutions candidates. Nous présentons ensuite une méthode de hashage pour supprimer les doublons de cet espace de recherche.

Nous exprimons de façon théorique la taille de l'espace de recherche et montrons qu'elle est diminuée significativement par nos méthodes. Nous effectuons des mesures de temps d'exécution et montrons que les résultats sont en accord avec l'étude théorique. Nous exposons quelques séquences intéressantes trouvées par notre algorithme et les implémentons dans les robots réels pour concrétiser notre approche.

Contents

Chapter 1

Introduction

1.1 Context of the Project

THE world of robotics has been studied intensively during the last few years. The aim of this research is to create machines that could replace or assist humans for some tasks, in which they could even surpass them.
The main limitation of traditional robots is their poor versatility and adaptability. This implies that the robots are always designed for a specific task. The problem is that, for some applications, both mission and environment are unpredictable. If we think, for example, about space exploration, the nature of the terrain is unknow before the robot has started the task. That is why it is very difficult to build a robot that is able to efficiently locomote on deep space planets, which explains why the current systems are so slow.

The problem of adaptive locomotion is indeed so hard that it has not been completely solved by nature. No animal is able to move in any environment efficiently. For example an albatross is really comfortable in the air because of the great size of its wings but not at all on the ground for the same reason. In order to bypass this limitation, robots should be able to select the best shape according any environment. This idea brought the research community to contemplate modular robots. These robots are composed of a set of similar units, modules, that can connect, disconnect and move around one another. For instance, imagine a robot composed of a chain of modules. This chain could first move like a snake, then connect its tail to its head to build a wheel shape, acquiring the ability to roll. If the terrain becomes too broken, the robot could make a run of connections and disconnections

to reconfigure as a spider in order to stride over rocks.

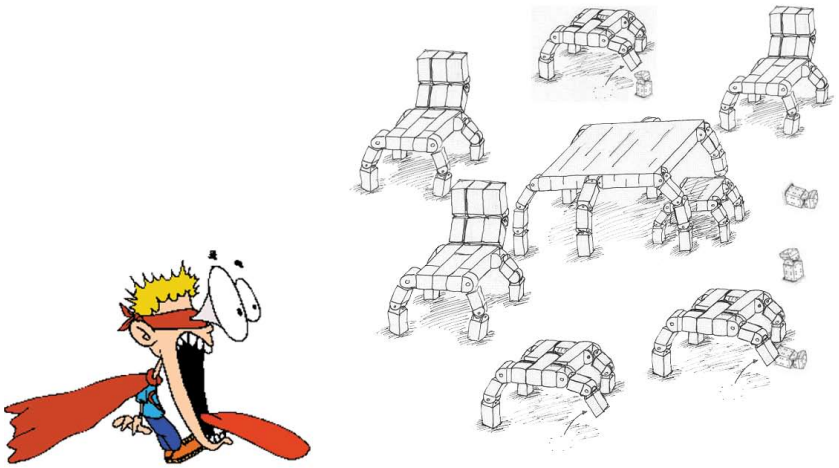


Figure 1.1: The future Roombots

The goal of our project is to design and control modular robots, called Roombots, to be used as building blocks for furniture that moves, self-assembles, self-reconfigures, and self-repairs. The type of scenario that we envision would be a group of Roombots that autonomously connect to each other to form different types of furniture, e.g. stools, chairs, sofas and tables, depending on user requirements. This furniture would change shape over time (e.g. a stool becoming a chair, a set of chairs becoming a sofa) as well as move using actuated joints to different locations (with or without a person sitting on it) depending on the users needs. When not needed, the group of modules can create a static structure such as a wall or a box to minimize its space occupation (see Figure 1.1).

The task of building such a modular robot is not easy though. In addition to the mechanical challenges it addresses two big problems : the **locomotion** of the whole structure and the **reconfiguration** from one shape to another

The first issue consists of how each module can set its position with respect to its neighbors in order to enable the locomotion. Due to the complexity of this task, the problem can be solved by using a neural network or a central pattern generator to control the moving

parts of the module, evolving their parameters. This first problem is not the purpose of this project.

The second issue is which module should be moved to which position in order to go from one configuration (shape) to another. Finding one sequence leading to the final configuration is proved to be a NP-complete problem, the number of possible configurations exploding exponentially with the number of modules. Due to the relatively high time consumption of the reconfiguration process of the real modules we can not satisfy ourself with the first acceptable reconfiguration sequence. The new problem is so to find the best sequence (i.e. the smallest number of moves) to reconfigure from an initial to a final configuration. This problem is the optimization problem corresponding to the former, thus NP-hard.

In this thesis we will first give the state of the art of the approaches to solve this problem, then we will explain our approach. Some experiments and results will be presented and discussed and guidelines for future work will be given in the last chapter.

1.2 Theoretical Background

HERE, we address the previous attempts to solve the reconfiguration problem explained before. This issue has been tackled by the scientific community several manners according to the type of robot, reconfiguration process or desired application.

The planning algorithms found in the literature fall into three main reconfiguration classes : **Substrate reconfiguration**, **Mobile reconfiguration** and **Closed-chain reconfiguration**.

Substrate reconfiguration designs are composed of modules which can only attach and detach at discrete locations of the configuration lattice. These modules (also called lattice type robots) can usually "roll" around one another to travel from one location to another. A *move* is then describe as a set of module disconnecting from one location, traveling along the surface of the lattice (which involves several connections and reconnections) to finally connect at its final location. See Figure 1.2 for an example of substrate reconfiguration.

Mobile reconfiguration hardware implementations are quite similar to substrate reconfiguration. They consist of modules that can, in addition to the moves allowed in substrate reconfiguration, move independently in the surrounding environment. A move would then consist of detaching from one position, moving in the environment (either on

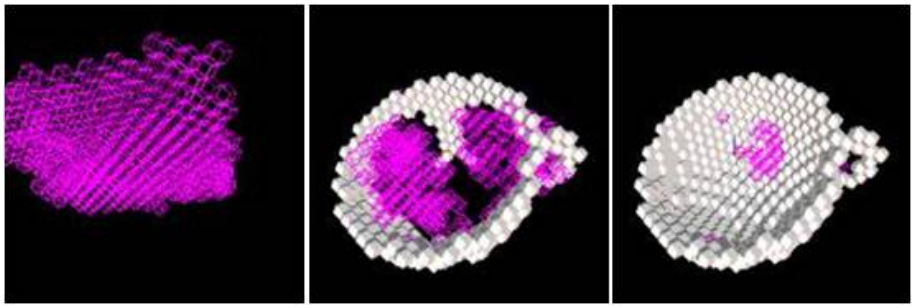


Figure 1.2: An example of substrate reconfiguration. Starting from a flat squared configuration (left), the moving modules (in pink) roll around the fixed ones (middle) to build a cup (right)

the surface of the robot or in the surrounding environment) and reattaching at another position.

Closed-chain reconfiguration is different from the two types described before. Those designs are composed of chains of modules viewed as one single entity. A chain can attach to other chains to build more elaborate structures or loops. Here the moves are much more complex since all modules of a chain have to collaborate to enable a movement. This last kind of reconfiguration involves only one connection/disconnection per move but might need the degrees of freedom of all the modules of the chain to bring one point of the chain in front of another chain or another module of the same chain and make a connection.

Another big distinction between the approaches in modular robotics is the location of the control. The two solutions are :

Centralized control : The reconfiguration planning and high-level movement control of the modules is made by a single machine which can communicate with the modules to give them orders. It has several advantages. First, control unit has no size limitation. It can be, for example, a normal computer or a super computer, which might reduce the time consumption of the system. The control unit has a global view of the whole configuration. This allows to have more information on the system such as the position of the modules in an absolute frame, leading to a simplified reconfiguration planning.

Decentralized (or distributed) control : Each module has its own computation unit and communicates only with the neighboring modules. In contrary to centralized control, **decentralized control** imposes that the control unit fits into the structure of the modules, as well as the motors, low-level controllers etc. The amount of computation

doable by one module is thus limited. Each module only has knowledge of its neighbors, which makes the planning more difficult. But distributed planning has a big advantage : it allows for autonomous control. There is no need for external control which might not be possible for long distance applications.

1.3 State of the Art

A good survey on modular robotics has been previously described in [1]. In this chapter we will focus on the state of the art of the reconfiguration algorithms for modular robots.

1.3.1 Substrate reconfiguration

Substrate reconfiguration is the type of reconfiguration that has been the most addressed in the literature. The two main approaches then are **centralized** and **decentralized** approaches.

Centralized approaches usually build the tree of all possible configurations using metrics and heuristics to guide the search. In [2], Pamecha et al. describe three useful metrics for reconfiguration of modular robots. The *Overlap Metric* considers the number of non overlapping modules. The *Minimal Number of Moves Metric* counts the smallest number of moves needed to reconfigure (which is not computational). The *Optimal Assignment Metric* allows an optimal relabeling of the modules of the initial and final configurations. Pamecha et al. provide the method to compute this *Optimal Assignment Metric* using the well know in networking science *Hungarian Algorithm*. Finally, they use this metric in an algorithm based on simulated annealing. This method is used in [3] and enhanced by using intermediate configurations to divide the problem into simpler ones. The algorithm is then applied to their hardware setup. It is also used by Durma et al. in [4] to build a holonic hand. Durma et al. use the adjacency matrix of the graph to find the distance (number of non common edges) between the initial and final representations. The same kind of approach is used by Nelson in his PhD thesis [5] but Nelson enhances it with kinematic constraints.

Decentralized approaches involve different techniques. In [6] an algorithm based on the *PacMan* game is used to plan the reconfiguration. After having computed the difference between the goal and initial configurations, “plan-pellets” are diffused from the goal positions through the modules. When those pellets meet a movable module they

change into "path-pellets" which this module just has to follow. This enables a fully parallel reconfiguration by using different pellets for different goal positions. This method is applied to the *Telecube* modules [7]. Støy, [8] uses gradient propagation through the structure to attract modules to their final position by having them climb the higher gradient to find the shortest path. Durma et al. [9], [10] use linear algebra applied to the neighborhood of a module to predict the spatial distribution of the whole configuration from the point of view of one module.

Bhat et al. [11] and Nguyen et. al. [12] use hierarchical planning. They group the modules into meta-modules, plan the reconfiguration at the meta-module level and complete it finally at the module level. The *M-TRAN* team [13] [14] describes a very inventive hardware setup. They use two-level planning : a global planner for cluster reconfiguration and a local planner for individual module movements.

Bojinov et al. [15] use simple local control rules (embedded in the modules) and limited communication to direct neighbors for emergence of complex shapes adapted to a specific functionality (without explicitly specifying the final configuration).

White et al. [16] have a completely different approach. Their modules evolve in a fluid and attract each other. The reconfiguration is done by detaching from the main structure, randomly traveling in the fluid and being attracted to another position.

1.3.2 Closed chain reconfiguration

The **closed chain reconfiguration** problem has been tackled much less by the scientific community. Casal et al [17] decompose the whole configuration not into modules but as a tree of chains and loops of modules. They propose two algorithms to perform the reconfiguration of *PolyBot* module configurations. One involves introducing a very simple intermediate configuration in the reconfiguration process. The other aims at matching the number of levels of the tree and then level by level the size and type of substructures. Their following work [18] introduces the concept of the *Robotic Connectivity Graph* with embedded properties to simplify the computation of these two algorithms. They prove that the reconfiguration process can be done in $O(\log n)$.

These works solve quite efficiently the reconfiguration problem for graph representation of chain type robots but none of them care about the kinematical feasibility (DOFs of the chains, collisions, torque limit) of the moves they plan.

Chapter 2

Our Approach

SINCE the Roombots project was a new project, the first thing was to quickly get a general idea of the main problems to be addressed and the possible solutions.

2.1 Preliminary study

2.1.1 Formalization of the problem

We separated the general problem into two main sub-problems :

- **Generating the valid moves** from any configuration, i.e. deciding which module can move and to which position. This allows us to generate the graph representing all the valid sequences of moves from one configuration.
- **Finding heuristics** to select the sequences more likely to bring us to a given final configuration.

To generate the valid moves we have to define the validity of one move. A move is said to be valid if :

- The move respects the **actuation capabilities** of the specified module i.e. a module can only move around its degree(s) of freedom.
- The move does not involve any **internal collision** between the moving unit and the surrounding modules.
- The move does not involve **disconnection** of the whole structure. The graph representing the configuration must stay connected.

To handle the second problem we also have to care about several issues. First of all, we need a way to compare a given configuration to the final one. For this we need a good metric characterizing the distance between the two configurations. This metric should reflect the cost according to some predefined criteria (e.g. number of connections/disconnection, spatial distance between the modules, number of moves around the degrees of freedom of the modules, maximum common subgraph). Then, we must find a way to traverse the configuration space using this metric as a guide. This could be a simple best-first-search, a probabilistic search, a backtracking algorithm etc.

We made a first investigation on these problems by implementing a very simplified simulation. We chose to make all our simulations in two dimensions because 3D simulation would only increase the size of the search space and induce the necessity of developing a more complex 3D interface without introducing new real problems. Thus, if our algorithm is efficient in 2D, it will also be efficient in 3D.

First of all the modules used here were simple squared modules which can roll around one another as shown in figure 2.1. Each module can move independently on the lattice surface of the configuration so the reconfiguration process can be classified in the substrate reconfiguration class. The computation of the valid moves for this set up is quite simple. We did not even consider all the collisions and tested the system with configurations where collisions could not occur (in fact most of the configurations for this set up). A module can move if and only if :

- It does not have neighbors on opposite faces (it avoids collisions between the neighboring modules)
- It does not have more than two neighbors (immediate from the last assertion)

The valid moves are represented in Figure 2.1.

2.1.2 Heuristic

The problem being well understood and formalized, we implemented a heuristic based on the *Manhattan Taxicab Metric*. The *Manhattan distance*, is computed as follows :

$$\forall M_1(x_1, y_1), \forall M_2(x_2, y_2), d_M(M1, M2) = |x_2 - x_1| + |y_2 - y_1|$$

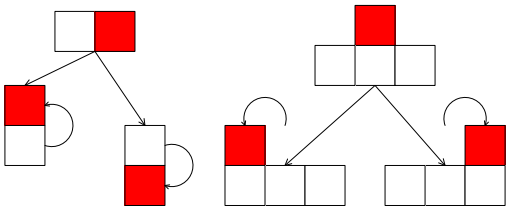


Figure 2.1: The valid moves of the simplified modules. Left : rolling around one module ; right : moving on a surface.

All the metrics based on the *Manhattan distance* suppose that the label of the modules (i, j) is made properly. A good way of optimally labeling the modules is to use the *Hungarian algorithm* which is the same as computing the *Optimal Assignment Metric* that is of complexity $O(n^3)$ [2]. Due to the relatively high complexity and difficulty of implementation of this last metric, we chose not to use it for this preliminary study in order to have very quick results. Instead, we implemented a very simple metric which is not optimal but happened to give correct results. This metric is computed using the distance matrix D between the current and final configurations. This matrix $D_{n\times n}$ is defined as follows :

$$D = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \text{ where } a_{ij} = d_M(i,j)$$

And $d_M(i,j)$ is the *Manhattan distance* between the modules i of a first configuration and j of a second configuration.

Then, we assign two modules whose distance is minimal to one another. The modules which do not move are automatically assigned to each other (i.e. if modules i and i' are at the same place, then $a_{ii'} = 0$). The results of the planning are thus better if we provide the maximum common subgraph of the two configurations.
The metric we used can be expressed this way : $\sum_{i=1}^{n} \min_{j=1..n} a_{ij}$, where j has not already been involved in any "min" expression.

We chose to represent the space search as a tree instead of a graph (or a lattice) because it is easier and less time consuming to construct : if we wanted to construct the lattice of configuration we would have to check if each new configuration has already been computed in another branch to make the link. So at each level we chose the best node according

to the previously defined metric and computed subsequent nodes. If the distance becomes zero it means we have reached the goal configuration. In order to avoid infinite loops we check for each new child if it is not already present in the current branch.

A sequence found by our algorithm to reconfigure from a "snake" to a "cross" is shown in Figure 2.2.

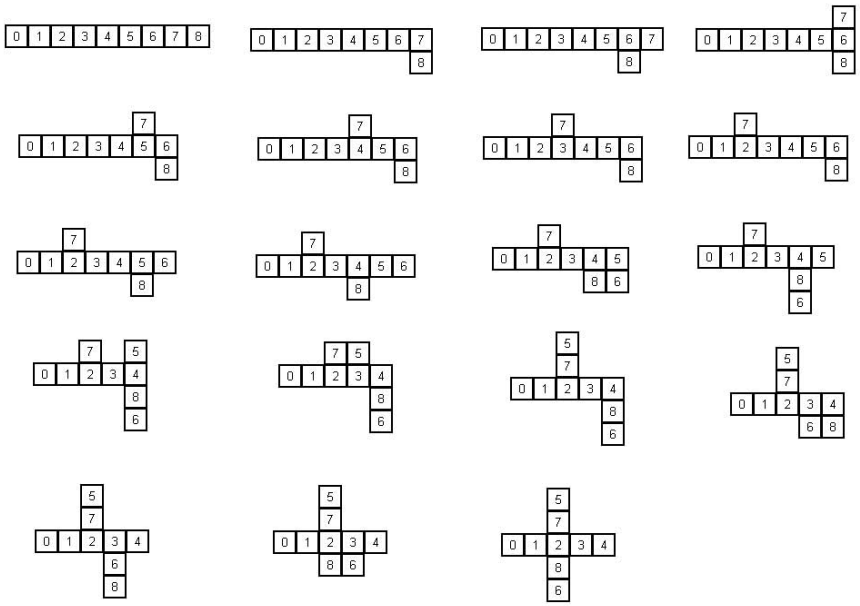


Figure 2.2: A sequence showing reconfiguration from a snake to a cross

2.1.3 Conclusions of the preliminary study

The main conclusions that we made from this preliminary study are the following. First, for our application the *Manhattan distance* between the modules is not a very accurate indication, even if we use the Optimal Assignment Metric. For instance, if one module is turning around a chain, the *Manhattan distance* to its final position may be decreasing rapidly without the chosen path being the best one. If we look at Figure 2.3, a gradient following method using the *Manhattan distance* would not find the optimal path. Indeed, the *Manhattan distance* on the first move of the red path is less than the one of the blue (optimal) path.

Thus we need to find something to enhance the efficiency of our metric when applying it to the real Roombots. The tree structure seems to be a good way to represent the search

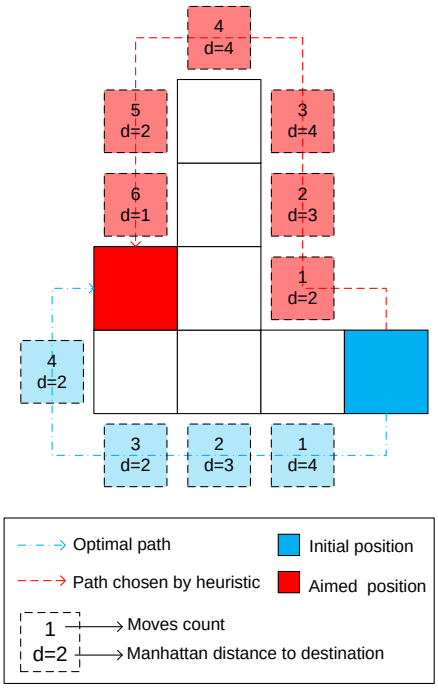


Figure 2.3: A case where the *Manhattan distance* is not an accurate metric

space, so we kept it. We will now try to adapt this general system to our real problem.

2.2 Reconfiguration of the Roombots

The Roombots, being devoted to have furniture-like shapes, will be composed mostly of chains and loops (or static structures). Furthermore, one Roombot cannot move by itself. Thus, we took a closed chain reconfiguration approach. Figure 2.4 represents 3D models of the modules that may be used for the Roombots.

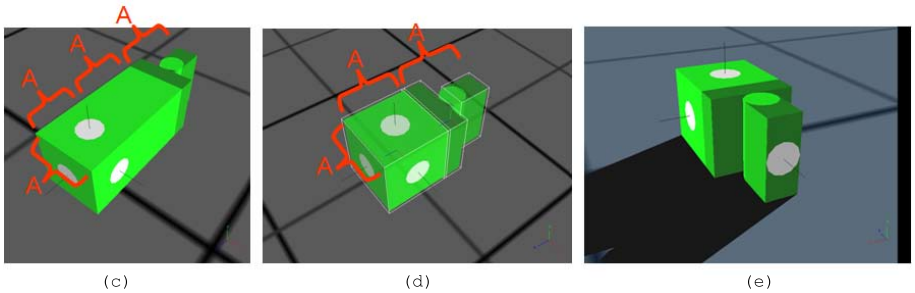


Figure 2.4: The 3D modes of (a) the Dof-Box II modules and (b) the Yamor modules.

2.2.1 Computation of the valid moves

The computation of the valid moves for Roombots configurations is a much harder task than for the preliminary study. We define a *move* as one **connection** or one **disconnection**. So this problem can be represented as a graph problem (see Figure 2.5), a move being defined as addition or deletion of an edge.

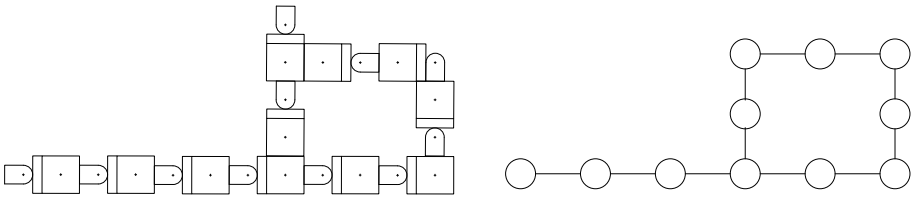


Figure 2.5: A robot configuration and its graph representation

If we look at this problem only as a graph theory problem, we might consider all the possible connections and disconnections but these moves might not necessarily be kinematically valid. What we want is to obtain the mechanically feasible connections/disconnections for the real modules described before and only those ones.

Getting all the feasible connections from one given configuration is not trivial since we have to first move the modules at contact. For this we considered moving all the servos in each of their possible positions. But as can be seen in Figure 2.6 the order of the movement is not to be neglected since some can induce collision.

This generates a tree of all the possible sequences of moves in order. For the rest of this thesis we will simply call this tree the *Servo Movements Tree* (see Figure 2.7 for an example)

At each level of the *Servo Movements Tree* we check two things :

- Does this move involve a collision ?
- Is it possible to reconnect two modules (add one edge to the graph representation) ?

For the first point we check the presence of modules on the trajectory of the moving subgraph. This can be done in $O(n)$. This trajectory defines a disk arc as represented in Figure 2.8. This figure also represents the positions that we check to see if a reconnection is possible. This check is also of complexity $O(n)$. The positions are checked for connector presence which are on the faces and on the servo of a module.

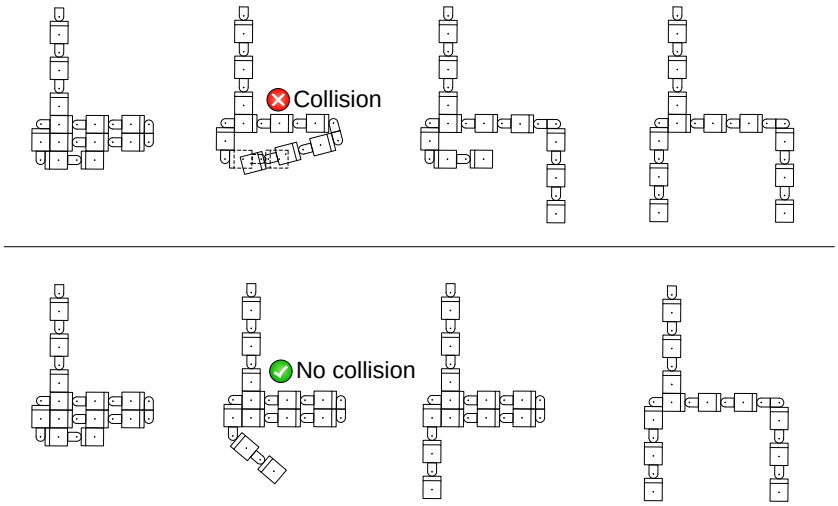


Figure 2.6: Two sequences involving the same servo movements but in different orders. In the first case (top) a collision happens and in the other one (bottom) the sequence is collision free.

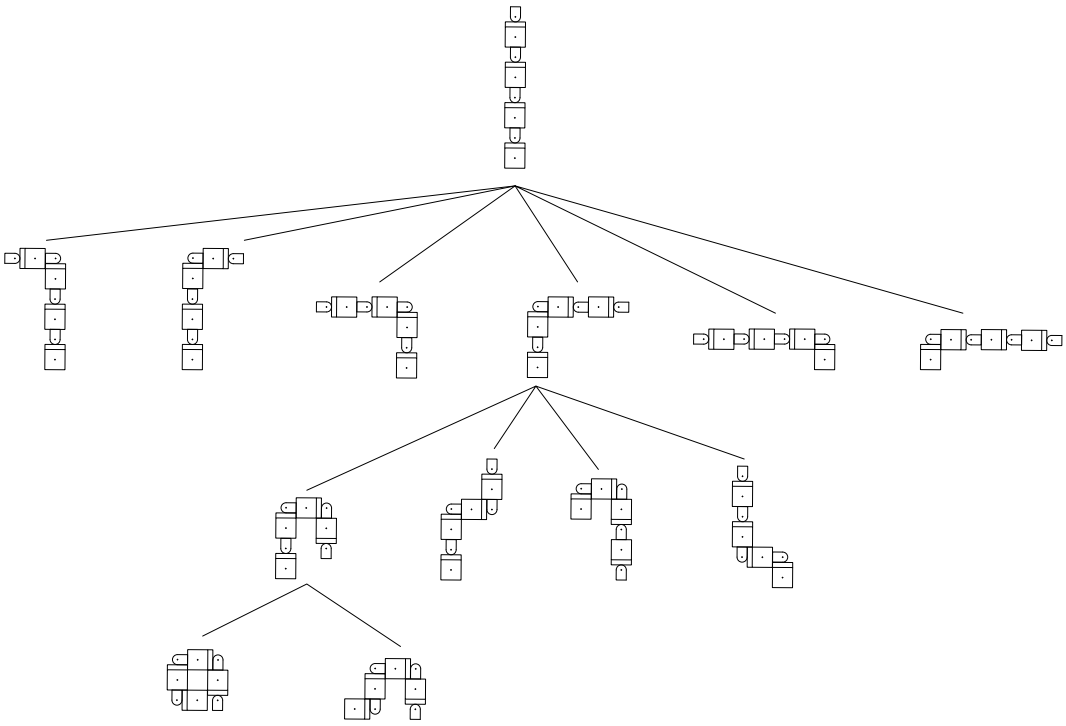


Figure 2.7: An example of the tree of all possible servo movements

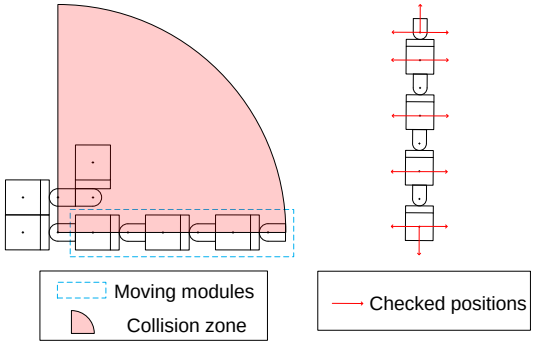


Figure 2.8: The checked space for collisions (left) and the checked positions for reconnection (right)

When a successful reconnection has been made, the involved configuration represents a new node of the tree of possible moves (which considers only the graph representation of the configurations; this is not the *Servo Movements Tree*). From now on we will call this second tree the *Moves Tree* since each child node is obtained from its parent by adding or removing an edge (making one connection or one disconnection). Figure 2.9 shows the same example as in Figure 2.11 (left) and its associated *Moves Tree* (right). The *Moves Tree* will be used to find a path from one configuration to another.

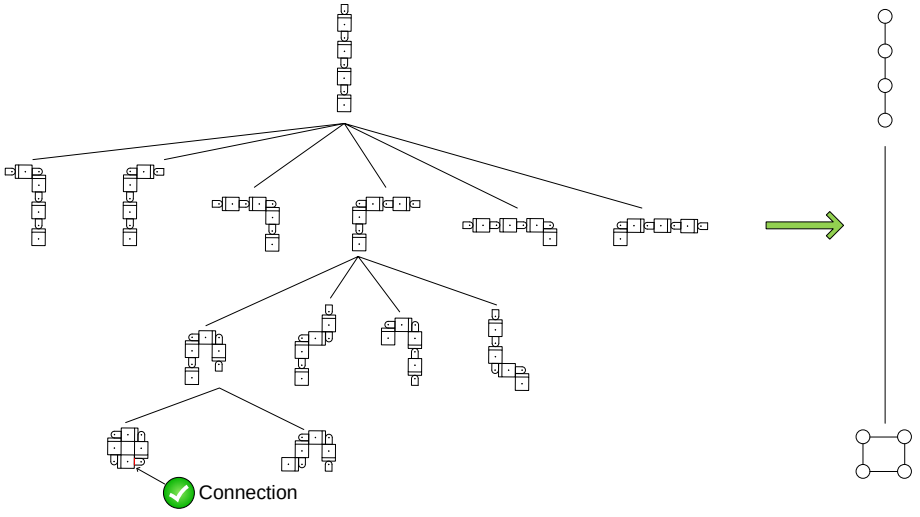


Figure 2.9: The correspondence between the *Servo Movements Tree* and the *Moves Tree* as defined by the graph problem. Only the configurations involving an addition (connection) or deletion (disconnection) of an edge are kept in the *Moves Tree*

For each level of this configurations tree we perform all the possible disconnections. Checking if a disconnection is possible does not involve any kinematics so we can perform this check on the graph representations. A disconnection is then possible if the graph stays connected after the deletion of an edge. To check this, we traverse any induced sub-tree of the graph. If the number of nodes in the tree is less than the one in the graph, the graph is disconnected (see Figure 2.10). This algorithm is of complexity $O(n)$.

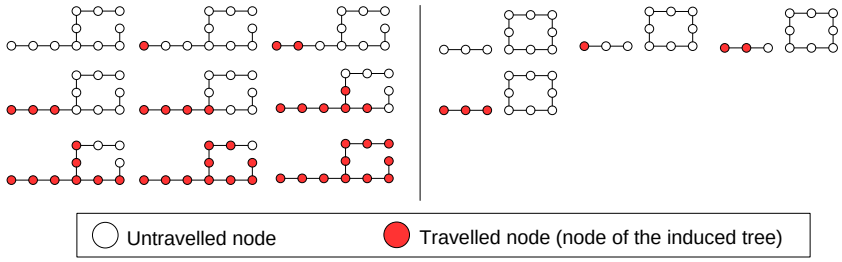


Figure 2.10: The disconnection check on a connected (left) and disconnected (right) graph.

Just like the possible connections, each possible disconnection is a child of the *Moves Tree*. Figure 2.11 represents the full *Moves Tree* of the configuration represented on the left. Before adding a child to this tree we check that the same graph representation is not already present in the same branch so that we do not create loops in our search (that is why there is only three leaves instead of four in Figure 2.11).

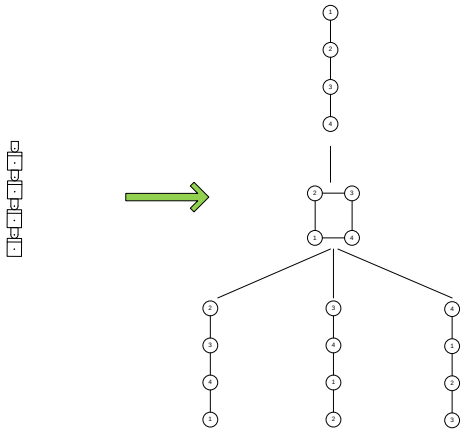


Figure 2.11: An example of a configuration and its corresponding *Moves Tree*

2.2.2 Complexity reduction

Now that we can generate the tree of all possible moves from one configuration, we can treat the initial problem (i.e. finding the best sequence of moves from one configuration to another) as a graph theory problem. But it is interesting to see the overall complexity of the process of obtaining this graph. If we go back to the tree of the servo movements in Figure 2.11 as an example, the number of configurations checked is :

1^{st} **level :** $2 \times 3 = 6$

2^{nd} **level :** $6 + 2 \times 2 = 10$

3^{rd} **level :** $10 + 2 \times 1 = 20$

Finally the number of checked configuration can be expressed this way (calling n the number of connected servos and not the number of modules, each servo having 2 positions to move to) :

$$N = 2n + 2^2 n(n-1) + \ldots + 2^n n! = \sum_{p=1}^{n} 2^p \frac{n!}{(n-p)!}$$

Under the sum, one can recognize the formula giving the number of permutations without repetition. This is understandable since we consider every movement of servo in order (permutation) but a servo can not be moved twice (without repetition).

This complexity is highly exponential so intractable for more than a few modules (connected servos). Consequently we have to find heuristics to reduce the search space.

The first heuristic that we imagined is based on the following observation : the probability to have a reconnection is inversely proportional to the sum of the distances between the modules of the configuration, i.e. reconfigurations are more likely to happen in configurations in which the modules are near to each other. Thus we decided to use this distance between the moving and fixed modules as a quality function of the movements and called it the *Module Distance Metric*.

The algorithm proceeds as follows : each servo movement gives us two new configurations ; for each of these two configurations we compute the *Module Distance Metric* and we keep the best one (cut the branch of the less desirable one). This process is described in Figure 2.12.

This heuristic enables us to divide by two the number of configurations checked **at each level**. Thus at the lowest level the complexity is divided by 2^n. The new complexity is

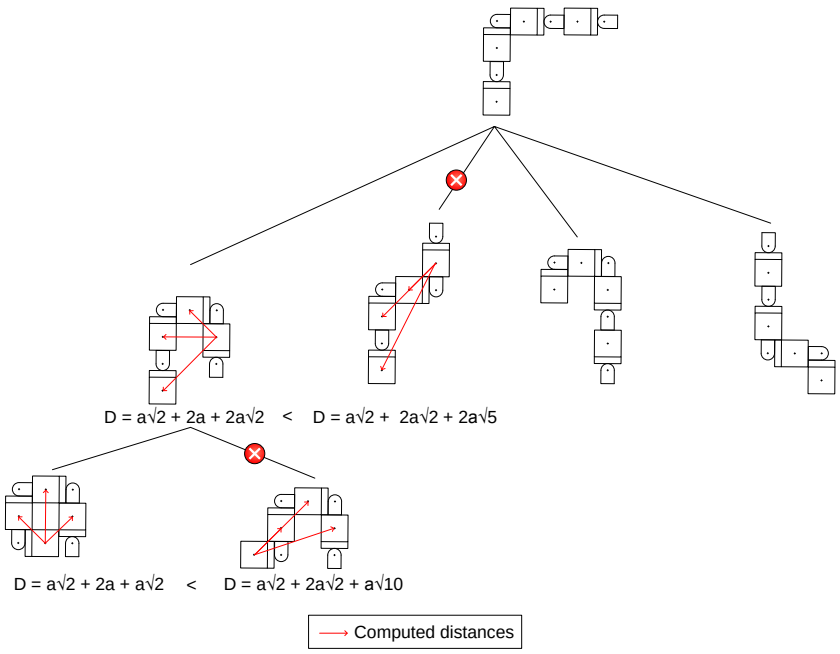


Figure 2.12: Illustration of the heuristic using the *Module Distance Metric*.

expressed as :

$$N' = n + n(n-1) + \ldots + n! = \sum_{p=1}^{n} \frac{n!}{(n-p)!}$$

This complexity is still exponential as can be seen in Section 3.1.1 on page 22. Thus we have to find other ways to reduce the search space.

To further reduce the search space, this we made the following observation. The order of the servo movements has to be considered as explained in Section 2.2.1 on page 12, but this introduces many duplicates in the *Servo Movements Tree*. Indeed when several orders of servo movements to the same configuration are collision free, this automatically introduces duplicates as can be seen in Figure 2.13.

As long as we have one valid sequence of servo movements we do not need to keep the other valid sequences. So those duplicates have to be removed from the configuration tree. To do this we want to keep a list of configurations obtained without collision and check if a configuration is already present in this list. In order to minimize the time consumption of this process we imagined a unique signature of a configuration that we store in a sorted list. As we are sure that the connectivity of the graph is the same between all the configurations checked we just need to store the angle of each servo. We give an ID to each servo angle (0 for 0 $^\circ$, 1 for -90 $^\circ$, 2 for +90 $^\circ$). Our signature is then simply a number in base 3, thus

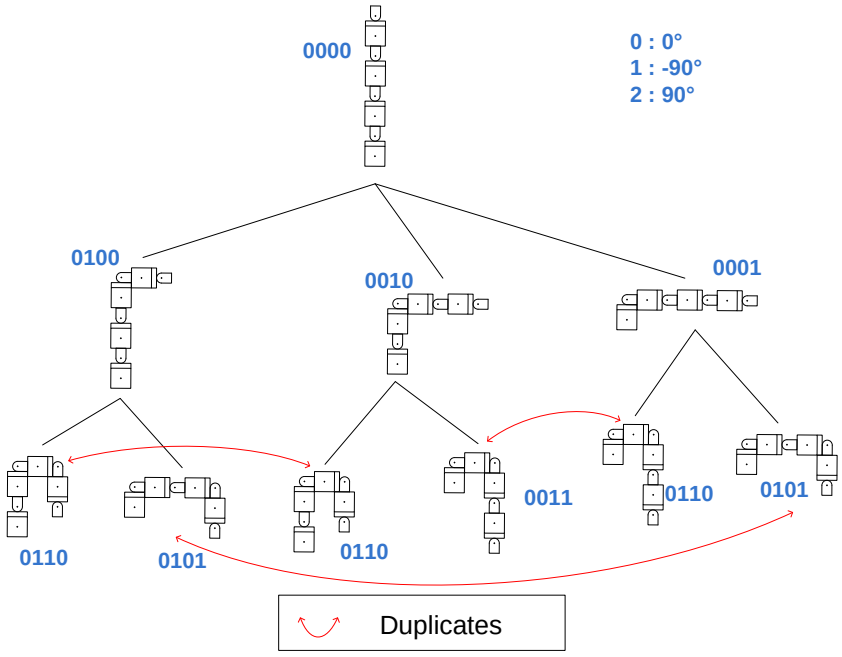


Figure 2.13: Duplicates in the *Servo Movements Tree*.

very easy to compare. As an example, the configuration represented in Figure 2.14 would have a signature of 012020 (with modules labeled from left to right).

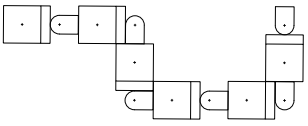


Figure 2.14: A randomly chosen configuration : its signature is 012020

The signature list being sorted, checking the presence of a signature in the list is done in $O(\log n)$.

The new complexity of this can also be expressed in the worst case scenario (no collision) so as an upper bound. When no collision occur, at each level of the *Servo Movements Tree*, we have every combination of servo movement, not considering the order since we keep only one valid order and without repetition. So at level p of this tree, the number of configurations is expressed as $\frac{n!}{p!(n-p)!}$.

This means the total number of configurations checked is $\sum_{p=1}^{n} \frac{n!}{p!(n-p)!}$. This value is also know in set theories as the *Power Set* of the set of servos. So the the total number of configurations checked can be simplified as the cardinal of the power set of the set of servos

thus :

$$N'' = 2^n - 1$$

The "- 1" is present because we do not consider the $\emptyset$ of the power set which would correspond to no movement.

2.3 Implementation

2.3.1 Graphical interface

First of all we developed a graphical interface to visualize the paths found by our algorithm (see Figure 2.15). This GUI represents the tree of configurations on the left. By clicking on a node, the 2D representation of the corresponding configuration is displayed on the right. The user can then generate the children of the current node. An option enables the user to generate the whole tree to a specified level. Finally, by right clicking on a node, the user is able to automatically send commands to the real robots to reconfigure it to the configuration corresponding to this node. This graphical interface has been developed in C++ .NET (managed C++) using Visual Studio 2005, the classes containing the real data (configurations, algorithms) being coded in native C++ for performance purposes.

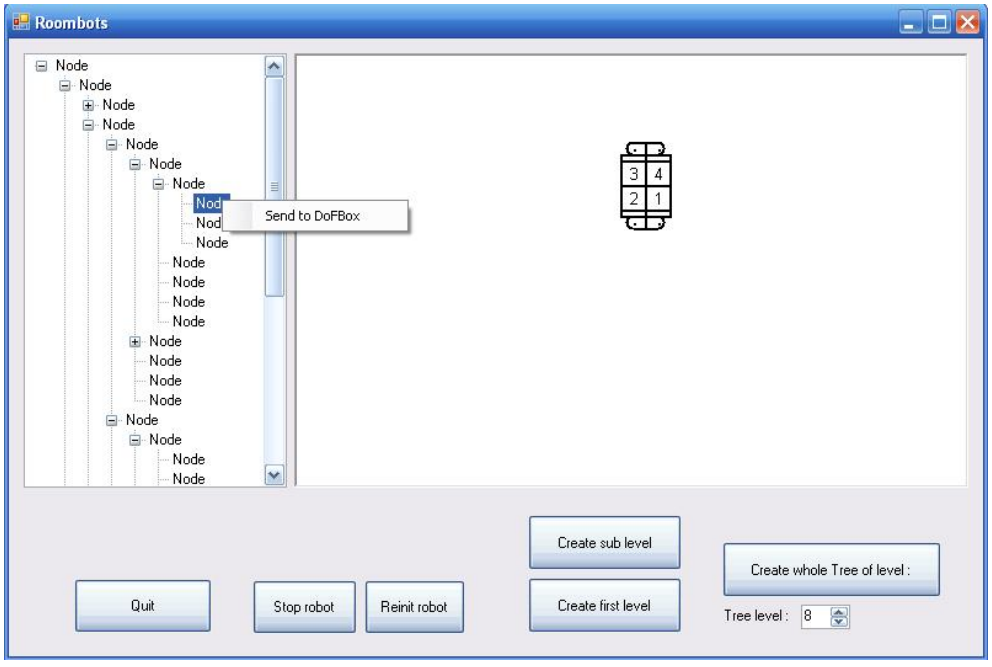
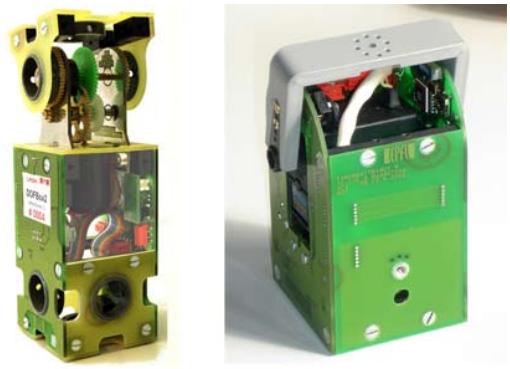


Figure 2.15: The graphical interface developed for the project.

2.3.2 Hardware implementation

The modules we used to materialize our approach are called the DOF-Box II. These modules have been developed by the Learning Algorithms and Systems Laboratory (LASA) [19]. The connection mechanism is still manual (screw) and the modules are interconnected by communication and power buses which prevents us from building up very complex configurations. The modules have one degree of freedom, one servo, as the ones we used for our implementation (Figure 2.4). Figure 2.16 represents a figure of the DOF-Box II (on the left) and the YAMOR (on the right) developed by the BIRG([20]) and which will be used as a basis for the final modules. Our Algorithm can be parametrized to adapt to the different dimensions of those modules. The DOF-Box II modules have six connectors (one on each face plus one on the servo) and can be linked to a computer by a serial cable + translator. The commands are sent to dedicated registers, interpreted by the processor which commands the motor. The motor activates a gearbox to gain couple. Figure 2.17 represents examples of configurations that can be built with the modified YAMOR modules.



(a) (b)

Figure 2.16: The modules used for the implementation. (a) The real DOF-Box2 modules - (b) the real YAMOR modules

The videos are available on the BIRG website [20].

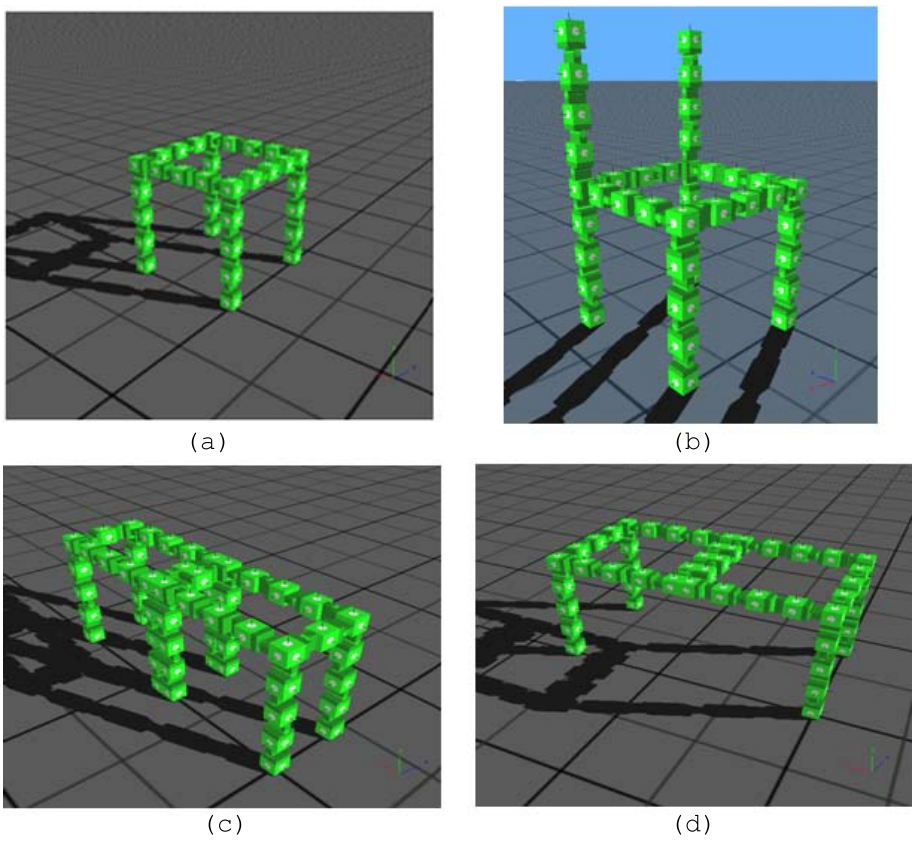


Figure 2.17: Some configuration that can be built with the modified YAMOR modules. (a) Stool - (b) Chair - (c) Bench - (d) Table

Chapter 3

Results and discussion

ALTHOUGH this thesis relates only the work of the author, thus one subpart of a bigger project, we are able to expose some results. In the first section we will present some results of our work and in the second one we will discuss those results.

3.1 Results

3.1.1 Complexity reduction

In this section we present the of results our methods to reduce the search space of our problem. These results could not be compared to other ones due to the two following points :

- The lack of data in the literature (which is explained by the second point).
- The fact that comparing the results of different hardware setups would not be relevant. Different modules imply different degree of freedom, thus different complexities of movement computation. The goals and specificities of the projects are usually different so are the specifications of the modules.

The graph in figure 3.1 represents the computation time (executed on a Pentium IV 2,8 GHz, 512MB RAM) with respect to the number of connected servos. The experiments have been made with a chain of four to ten modules which is one of the worst cases for our application since very few collisions occur and thus few branches of the movements tree can be cut. The torque limitation (maximum size of the moving subgraph) is set to four modules which is quite realistic according to the hardware specialists. The first curve represents this computation time when simply testing all the possible moves, the second one

using the *Module Distance Metric* as described in Section 2.2.2 on page 16. One can clearly see that in both cases the complexity is exponential (linear representation on a logarithmic scale) but with a much smaller factor in the case where we used the heuristic. For example for a chain of ten modules (nine connected servos) the computation takes about 23 minutes for the version when using the *Module Distance Metric* while it takes more than 22 hours without.

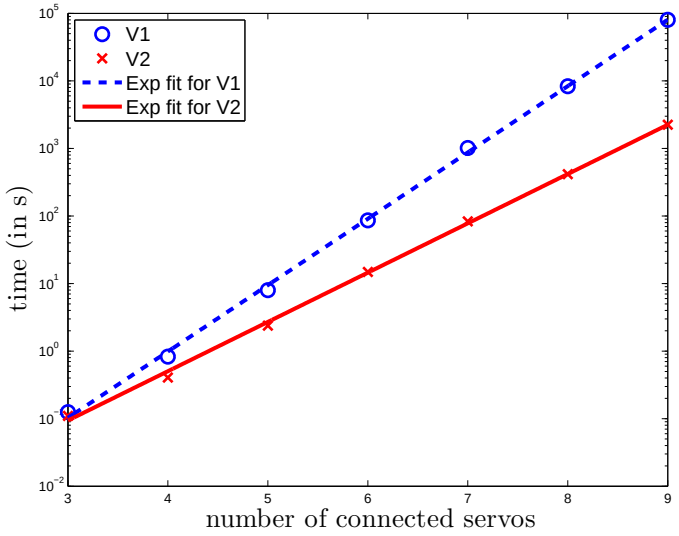


Figure 3.1: The computation times measured for chains of up to ten modules (nine connected servos) with (V1) and without (V2) using the *Module Distance Metric*. The scale is logarithmic and the measured values have been fit with exponential models

The graph in Figure 3.2 presents the complexity of the algorithm with and without the duplicate check presented in Section 2.2.2 on page 17. Again the complexity is decreased significantly. For example the computation of the valid moves for a chain of ten modules (nine connected servos) takes now a bit less than 5 seconds while it takes about 23 minutes when not removing the duplicates. We get about the same computation times with fourteen connected servos using the new version of our algorithm as with eight connected servos using the previous one.

As a matter of comparison, Figure 3.3 represents the theoretical sizes of the search space. One can note that the general shape of those three curves is similar to the ones representing the computation times in Figure 3.1 and Figure 3.2. C1 and C2 are pretty much strait lines while the slope of C3 is decreasing slightly on logarithmic scale since $2^n < e^n$. Thus the experimental results and the theoretical study are in accord.

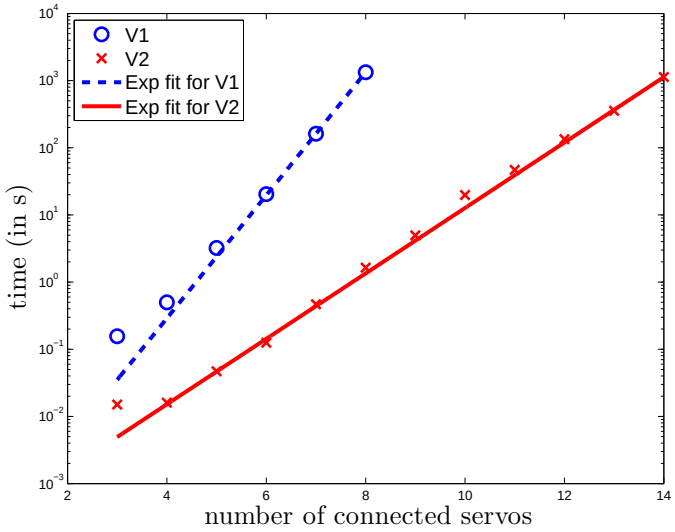


Figure 3.2: The computation times measured for chains with up to fourteen connected servos (using the *Module Distance Metric*) with (V1) and not without (V2) duplicates removal. The scale is logarithmic and the measured values have been fit with exponential models

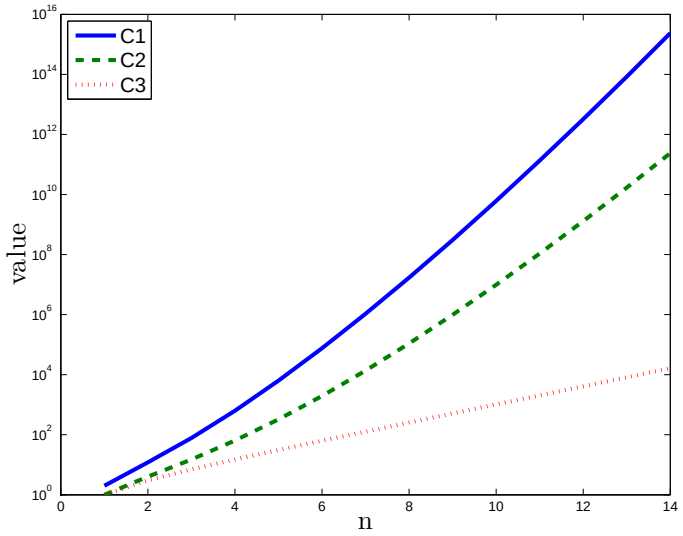


Figure 3.3: The theoretical sizes of the search spaces given in section 2.2.2 on page 16. C1 : $\sum_{p=1}^{n} 2^p \frac{n!}{(n-p)!}$, C2 : $\sum_{p=1}^{n} \frac{n!}{(n-p)!}$, C3 : $2^n - 1$

3.1.2 Sequences found

Although the computation time can be quite high sometimes as explained before, the software finally finds correct sequences of moves.

Figure 3.4 represents screenshots made at each step of the reconfiguration from a “snake” to a “chair”.

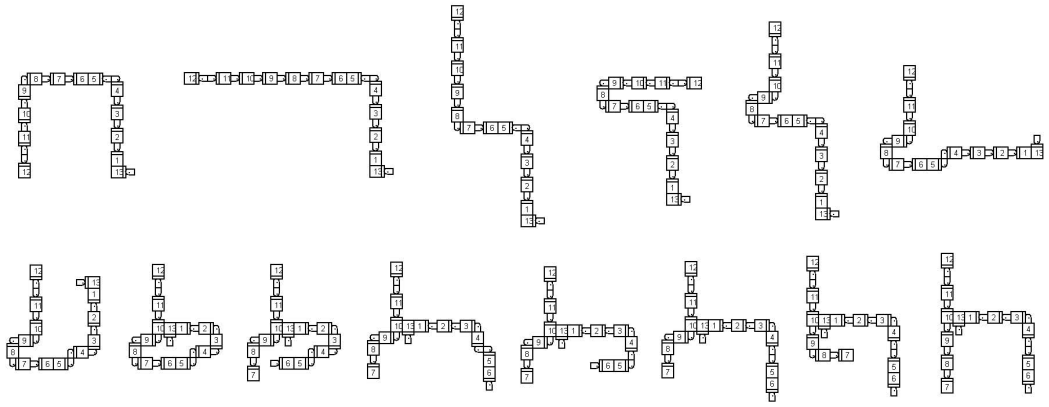


Figure 3.4: An example of reconfiguration sequence found by the software. From a “stool” to a “chair”

3.1.3 Hardware results

The following pictures are screenshots of the videos made for two reconfiguration sequences using the real modules. The sequences are quite simple due to the difficulty of building complex structures and to disconnect/reconnect modules with this setup. Figure 3.5 shows a ”snake” connecting its two ends to reconfigure to a “wheel” shape. Figure 3.6 another arbitrary chosen sequence containing connections and disconnections. Here those connections and disconnections are made manually and cut during editing.

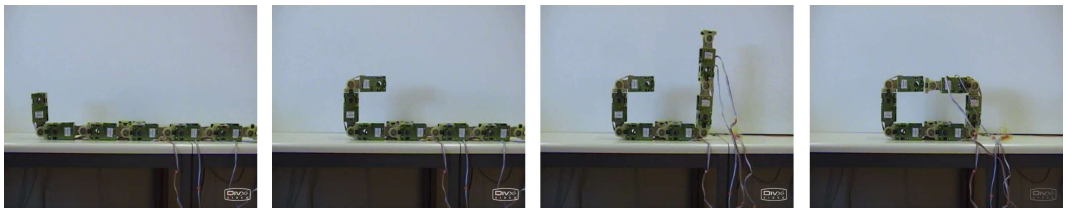


Figure 3.5: Reconfiguration of the DOF-Box : from a “snake” to a “wheel”

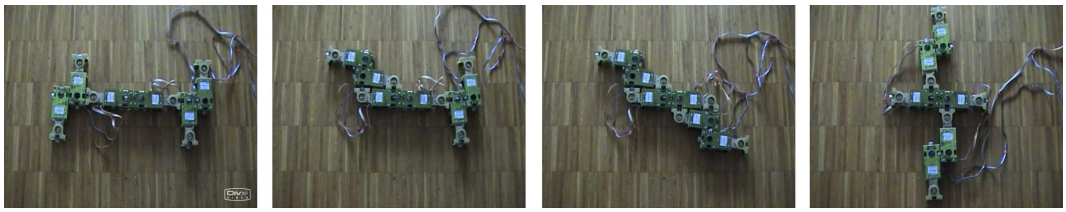


Figure 3.6: Reconfiguration of the DOF-Box with connection/disconnection : from “H” to “cross”

3.2 Discussion

As shown in Section 3.1 results our algorithm successfully finds interesting sequences of valid moves. Those sequences are exempt of collision and disconnection and are kinematically valid. The feasibility of the moves have been shown on real cases by the implementation on the real system. Those cases are really basic though since our hardware setup is very constrained for the moment. The control of the real robots is really simple for the user since he only needs one click to make the robot automatically reconfigure to one chosen configuration.

The complexity of the system has been reduced significantly : what took more than two hours to compute using a brute force check takes now about 45 seconds and configurations of more than a few dozen of modules should be tractable in reasonable time. But we have shown in 2.2.2 on page 18 that the size of the search space is still exponential ($O(2^n)$). The lack of data in the literature and the fact that there are very few relationships between the spacial repartition of the modules and the graph distance to final configuration make it really difficult to find heuristics for this problem. However, for our application (ie : furniture composed of a limited number of modules) this reduction of complexity may be sufficient.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

In this thesis we tackled the problem of generating the tree of valid moves that one modular robot configuration can perform. This problem in itself is NP-Hard and the size of the search space is highly exponential. We introduced methods to verify the validity of one move (i.e. kinematical feasibility, absence of collision and disconnection).

By implementing the found sequences into the module hardware, their validity was shown for real cases.

We expressed the theoretical size of the search space and introduced a metric based on the sum of the distances between the modules. We showed that this metric used in a standard best-first-search algorithm decreases the size of the search space by a 2^n order of magnitude.

We introduced a hashing method to remove the duplicates of the search space and shown that the complexity is reduced of a high factor, although it is still exponential.

4.2 Future Work

If the time limits of the project are not too strict, this section could be used as guideline for improvement of the current system.

We chose to consider the Roombots as chain type robots which makes the reconfiguration problem much more difficult. But chain type robots also have many advantages : The movement of full chains instead of single modules makes the reconfiguration much faster

since fewer moves are required to change from one shape to another. However, those advantages may not be exploited enough. Of course we generate all the valid moves so full chain movements are included, but many moves (i.e. connections and disconnections) are not relevant and thus involve unnecessary time consumption. To solve this imperfection we may find a way to force full chains that are present in initial and final configurations to stay connected as chains.

This could be done by a pattern detection algorithm which would detect chains and loops and compare them in initial and final configurations. Casal and Yim. present a similar approach in [17] and [18]. This work is explained in Section 1.3.2. After being detected the substructures would stay connected and move according to their DOFs. The rest of the structure that has not been matched by any pattern would be treated as it is now.

A slightly different approach is taken by the M-TRAN team in [13] and [14]. The reconfiguration is made at two levels : a *Global Planner* plans the reconfiguration for clusters of modules and a *Local Planner* for module scale planning.

To investigate this "substructure approach" further, we have looked for relevant substructures. Figure 4.1 shows an example of a possible substructure shaped as a wheel. This substructure can "roll" on the surface of the configuration and reconnects every 2A (A being the dimension of a side of the modules, see Figure 2.4 for precise dimensions.

Figure 4.2 shows another example of substructure composed of two modules. This substructure can connect every A.

These two substructures have different constraints on the surface on which it moves. The substructure in Figure 4.2 can not move on the surface in Figure 4.2. Even more efficient substructures can be found as the one in Figure 4.3 but they involve even more constraints on the surface. These constraints on the surface should be minimized with the new hardware setup that is currently in development. These new modules should have perfect cubic dimensions including the servo so there would be connectors every A on the surface of any configuration, independently of the way to modules are assembled.

When the relevant substructures have been chosen, they can be considered as single independent modules with their own moving abilities, trajectory (for collision check) and constraints. The movements of modules inside the substructure necessary to perform the movement of the whole substructure are precomputed. Thus the control of the substructures could be made at a higher level. For example the command : Substructure.Move(position)

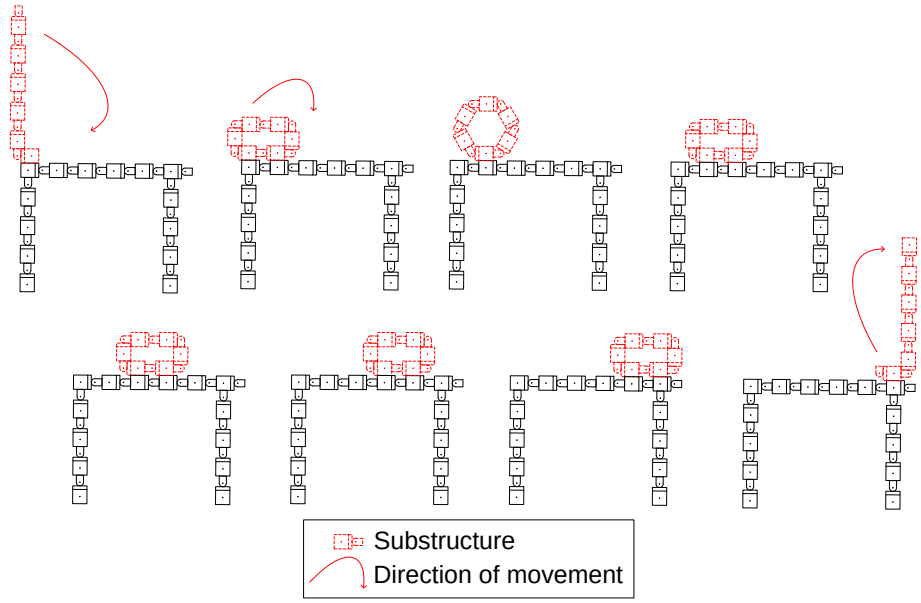


Figure 4.1: A possible substructure : the "wheel" which can "roll" on the surface and connects every 2A

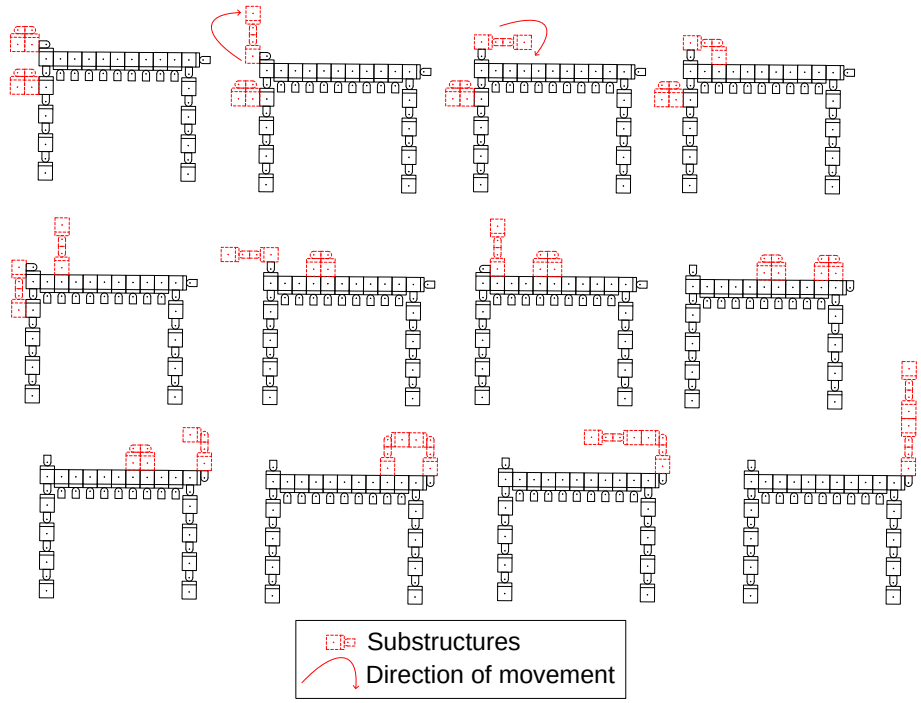


Figure 4.2: A possible substructure : two modules that can connect every A

would make a substructure move (considering that the position passed as argument is reachable) without regard to the movement of each module.

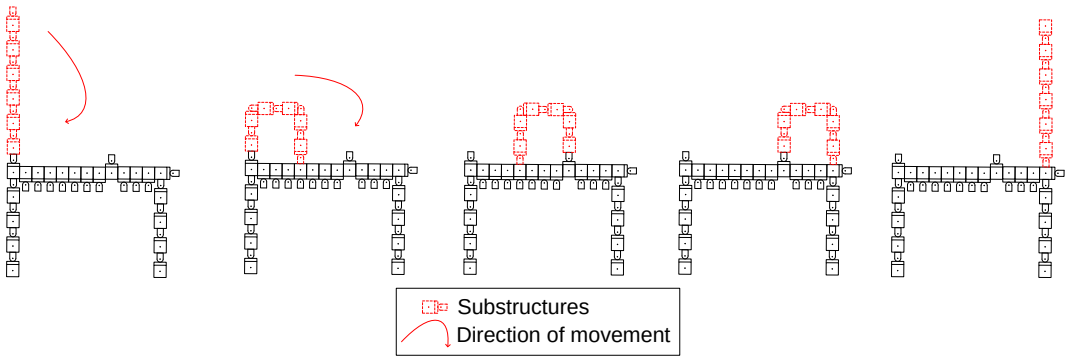


Figure 4.3: A possible substructure : two modules that can connect every A

When we look at Figure 4.2 we see that it would be interesting to have several substructures moving at the same time. This can be achieved by distributing, at least partially, the control of the reconfiguration. A possible implementation would be a multi-agent system where each substructure would be an agent, knowing its possible moving abilities and communicating with its neighbors to know, for example, the characteristics of other agents, the structure of the surface etc. This multi-agents approach is made possible by the fact that the substructures can move independently which single modules can not. The M-TRAN team has already developed a distributed control for their modules. Videos of distributed reconfigurations are available on the M-TRAN website [21].

Distributed control solves the problem of the exponential computation time by sharing the computation load between all the modules. Each substructure plans its own movements taking into account only its neighbors (direct or not). Thus the complexity of the planning for each substructure is not dependent on the number of modules of the whole configuration.

Of course the first thing to do, particularly if the time line is reduced, is to finish the development of the new module hardware and adapt our algorithm to it, but we are convinced that the previous guidelines are promising.

Chapter 5

Appendix

In this appendix, we give some pseudo-code of our algorithm to better explain the linking of the different parts of our work and between our work and the rest of the project.

Figure 5.1 presents the main parts of the whole algorithm to which our project belongs. The parts belonging to our work are highlighted to better show the linking with the rest of the project.

Figure 5.2 presents the details of the *Compute Children Nodes* procedure.

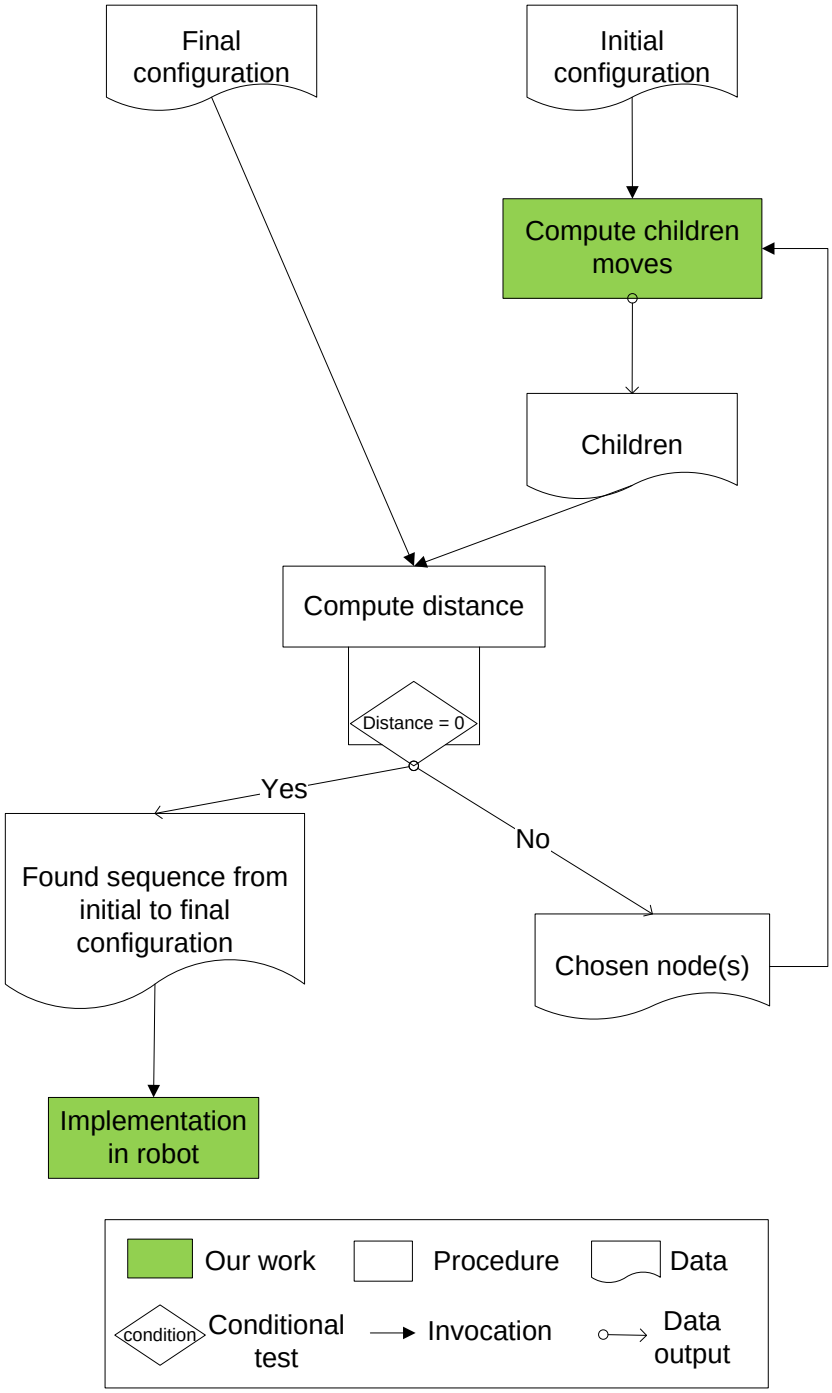


Figure 5.1: Illustration of the sequence of the full algorithm.

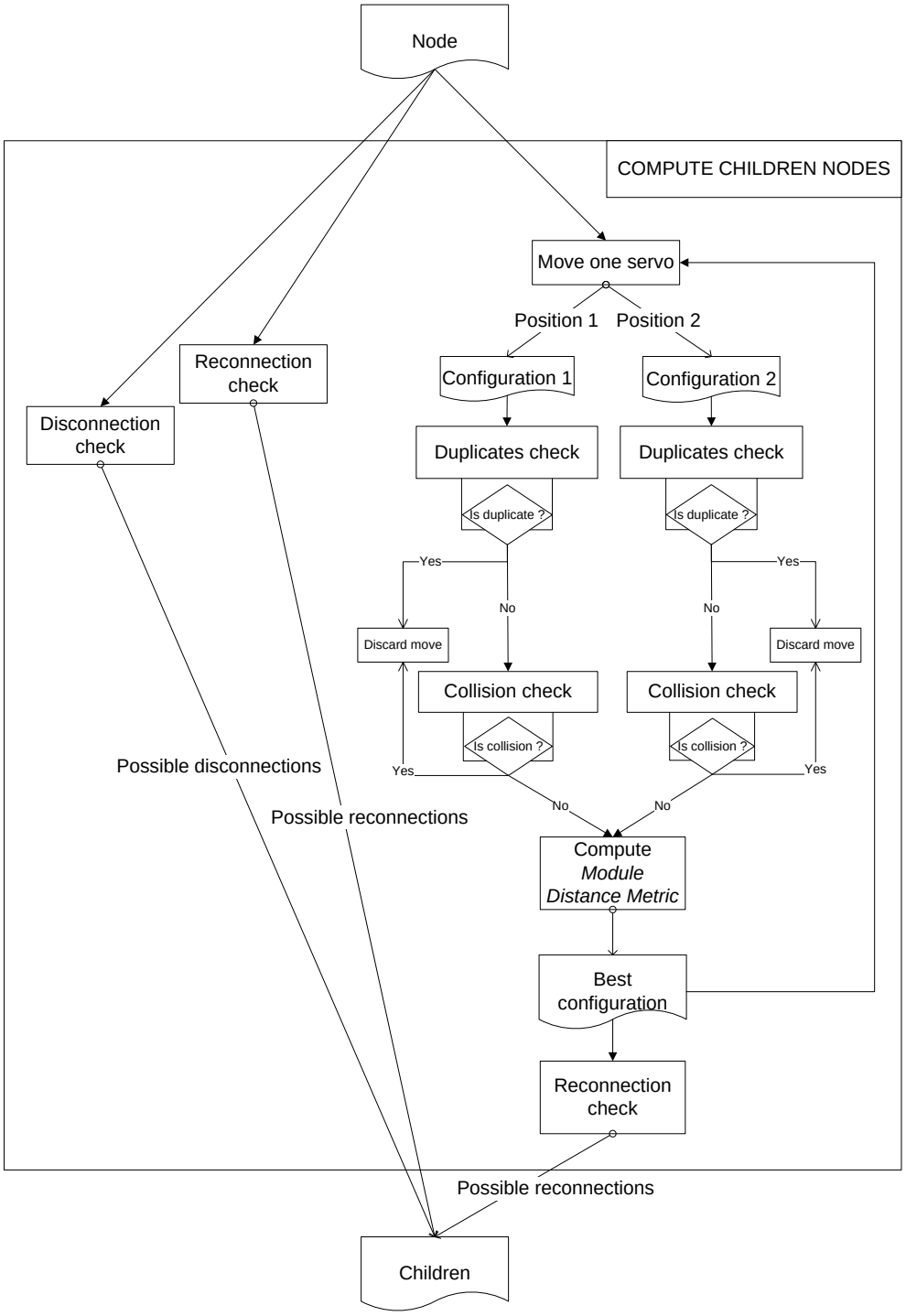


Figure 5.2: Detail of the *Compute Children Nodes* procedure

List of Figures

References

[1] S. Slee. A survey of motion planning for self-reconfigurable robots, December 2005. 5

[2] A. Pamecha, I. Ebert-Uphoff, and G. S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13:531–545, 1997. 5, 9

[3] C. J. Chiang and G. S. Chirikjian. Modular robot motion planning using similarity metrics. *Auton. Robots*, 10(1):91–106, 2001. 5

[4] M. Durna, A.M. Erkmen, and I. Erkmen. The self-reconfiguration of a holonic hand: The holonic regrasp. *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:1993–1998, 2000. 5

[5] C. A. Nelson. *A Framework for Self-Reconfiguration Planning for Unit-Modular Robots.* PhD thesis, Purdue University, May 2005. 5

[6] Z. Butler and D. Rus. Distributed motion planning for 3-d modular robots with unit-compressible modules. *Workshop on the Algorithmic Foundations of Robotics*, 2002. 5

[7] S. Vassilvitskii, M. H. Yim, and J. W. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1:117–122, 2002. 6

[8] K. Støy. Controlling self-reconfiguration using cellular automata and gradients. *Proceedings of the The 8th Conference on Intelligent Autonomous Systems (IAS-8)*, pages 693–702, 2004. 6

[9] M. Durna, A.M. Erkmen, and I. Erkmen. Self-localization of a holon in the reconfiguration task space of a robotic colony. *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA)*, 2:1748–1754, 2000. 6

[10] M. Durna, A.M. Erkmen, and I. Erkmen. Self-reconfiguration in task space of a holonic structure. *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2366–2373, 2000. 6

[11] P. Bhat, J. Kuffner, S. Goldstein, and S. Srinivasa. Hierarchical motion planning for self-reconfigurable modular robots. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 886 – 891, 2006. 6

[12] A. T. Nguyen, L. Guibas, and M. H. Yim. Controlled module density helps reconfiguration planning. *Workshop on the Algorithmic Foundations of Robotics*, March 2000. 6

[13] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji. A self-reconfigurable modular robot: Reconfiguration planning and experiments. *The International Journal of Robotics Research*, 21(10-11):903–915, 2002. 6, 28

[14] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7:431–441, 2002. 6, 28

[15] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (IRCA)*, 2:1734–1741, 2000. 6

[16] P. White, V. Zykov, J. Bongard, and H. Lipson. Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005. 6

[17] A. Casal and M. H. Yim. Self-reconfiguration planning for a class of modular robots. In G. T. McKee and P. S. Schenker, editors, *Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 246–257, aug 1999. 6, 28

[18] M. H. Yim, D. Goldberg, and A. Casal. Connectivity planning for closed-chain reconfiguration. *Sensor Fusion and Decentralized Control in Robotic Systems*, 4196:402–412, October 2000. 6, 28

[19] LASA (Laboratoire d'Algorithmes et Systmes d'Apprentissage) website. http://lasa.epfl.ch/. 20

[20] BIRG (Biologically Inspired Robotic Group) website. http://birg.epfl.ch. 20

[21] Videos of reconfiuration of the M-TRAN III modules on the M-TRAN website. http://unit.aist.go.jp/is/dsysd/mtran3/FlashMovie/mtran3/movie.htm. 30